УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Операционные системы»

Лабораторная работа №2

Вариант syscall: pci_dev, syscall_info

Студент

Иванов Е. Д.

P33111

Преподаватель

Барсуков И. А.

Санкт-Петербург, 2022 г.

Описание задания

Разработать комплекс программ на пользовательском уровне и уровне ярда, который собирает информацию на стороне ядра и передает информацию на уровень пользователя, и выводит ее в удобном для чтения человеком виде. Программа на уровне пользователя получает на вход
аргумент(ы) командной строки (не адрес!), позволяющие идентифицировать из системных таблиц необходимый путь до целевой структуры, осуществляет передачу на уровень ядра, получает информацию из данной структуры и распечатывает структуру в стандартный вывод. Загружаемый модуль ядра принимает запрос через указанный в задании интерфейс, определяет путь до целевой структуры по переданному запросу и возвращает результат на уровень пользователя.

Вариант: syscall -> pci_dev, syscall_info

Коды системных вызовов:

pci_dev:

```
#include “linux/kernel.h”
#include “linux/syscalls.h”
#include “linux/pci.h”

struct pci_dev* dev;
SYSCALL_DEFINE0(pci_dev)
{
	while ((dev = pci_get_device(PCI_ANY_ID, PCI_ANY_ID, dev)))
	{
		printf(KERN_INFO “pci found [%d]\n”, dev->device);
	}
	return 0;
}
```

ПРОГРАММА ПРОВЕРКИ:

```
#include “linux/module.h”
#include “linux/kernel.h”
#include “linux/unistd.h”
#include “sys/syscall.h”

int main()
{
	int ans = 0;
	ans = syscall(NUMBER_OF_SYSCALL);
	return 0;
}
```

РЕЗУЛЬТАТ:

```
[  206.776096] pci found [10688]
[  206.776101] pci found [4369]
[  206.776103] pci found [4097]
[  206.776104] pci found [4101]
[  206.776105] pci found [4096]
[  206.776107] pci found [4097]
[  206.776108] pci found [10548]
[  206.776109] pci found [10549]
[  206.776111] pci found [10550]
[  206.776112] pci found [10554]
[  206.776113] pci found [10520]
[  206.776114] pci found [10530]
[  206.776115] pci found [10544]
[  237.158794] pci found [10688]
[  237.158802] pci found [4369]
[  237.158803] pci found [4097]
[  237.158804] pci found [4101]
[  237.158805] pci found [4096]
[  237.158807] pci found [4097]
[  237.158808] pci found [10548]
[  237.158809] pci found [10549]
[  237.158810] pci found [10550]
[  237.158812] pci found [10554]
[  237.158813] pci found [10520]
[  237.158814] pci found [10530]
[  237.158815] pci found [10544]
```

syscall_info:

```
#include <linux/ptrace.h>
#include <linux/pid.h>
#include <linux/syscalls.h>

static struct task_struct *task;

SYSCALL_DEFINE2(get_syscall_info, int, proc_pid, void*, target) {
   int status;
   struct syscall_info *sc_info;

   task = get_pid_task(find_get_pid(proc_pid), PIDTYPE_PID);
   if (!task) {
         printk("Invalid PID: %d", proc_pid);
         return -1;
   }

   sc_info = kmalloc(sizeof(struct syscall_info), GFP_KERNEL);
   status = task_current_syscall(task, sc_info);
   if (status == 0) {
         memcpy(target, &(sc_info->data), sizeof(struct seccomp_data));
         kfree(sc_info);
         return 0;
   }
   kfree(sc_info);
   printk("Task %d is not blocked", proc_pid);
   return -1;
}
```

ПРОГРАММА ПРОВЕРКИ:

```
#include <sys/syscall.h>
#include <linux/seccomp.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char* argv[]) {

    if (argc < 2) {
      printf("Usage: ./test_sc <process_pid>\n");
      return -1;
    }

    int pid = atoi(argv[1]);

    if (!pid) {
      printf("Invalid PID: %s\n", argv[1]);
      return -1;
    }

    struct seccomp_data sc_data;
    int status = syscall(NUMBER_OF_SYSCALL, pid, &sc_data);
    if (status != 0) {
      printf("Something went wrong. See dmesg for more info\n");
      return -1;
    }

    printf("----- SYSCALL_INFO START -----\n");
    printf("Syscall number: %d\n", sc_data.nr);
    printf("Instruction pointer: %p\n", sc_data.instruction_pointer);
    for (int i = 0; i < 6; i++) {
      printf("Syscall arg %d: %lld\n", i, sc_data.args[i]);
    }
    printf("----- SYSCALL_INFO END -----\n");
    return 0;
}
```

РЕЗУЛЬТАТ:

Syscall number: 61
Instruction pointer: 0x7f7a672ae45a
Syscall arg 0: 4294967295
Syscall arg 1: 140720999802480
Syscall arg 2: 10
Syscall arg 3: 0
Syscall arg 4: 0
Syscall arg 5: 0

Ход выполнения работы:
Написать сисколлы в каталоге с ядром
Добавить линковку в include/linux/syscalls.h (asmlinkage)
Добавить в таблицу системных вызовов в arch/x86/entry/syscalls/syscall_64.tbl
Пересобрать ядро и перезагрузить машину

Выводы:
При выполнении лабораторной работы я познакомился с исходниками ядра Linux и системой системных вызовов. Понял как пересобрать ядро и добавить в него свои функции. Порылся в исходниках и понял куда передаются всякие структуры и за что они отвечают.